SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

KRAV

Overview

Project Summary

- Name: Krav
- Platform: Base
- Language: Solidity
- Repository:
  - https://github.com/kravmaxx/krav-contracts-aduit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Krav
Version	v1
Type	Solidity
Dates	Mar 15 2024
Logs	Mar 15 2024

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	3
Total Low-Severity issues	5
Total informational issues	4
Total	13

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	User funds may be locked in the contract	High	Business Logic	Pending
2	Cooldown mechanism may be bypassed via token transfer	Medium	Business Logic	Pending
3	Chainlink’s latestRoundData() might return stale results	Medium	Data Validation	Pending
4	Setting maxOpenInterestDai to 0 will not pause trading on a specific pair	Medium	Business Logic	Pending
5	Lack of duplicate checks for nodes	Low	Data Validation	Pending
6	Improper LINK fee for NFT order execution	Low	Business Logic	Pending
7	Inaccurate funding/rollover fee calculation	Low	Numerics	Pending
8	Lack of input parameter checks in the setPairParams() function	Low	Data Validation	Pending
9	Unimplemented functions	Low	Business Logic	Pending
10	Share price changes may cause the call to withdraw() to fail	Informational	Business Logic	Pending
11	Inconsistency between documentation and implementation	Informational	Inconsistency	Pending
12	Redundant code	Informational	Redundancy	Pending
13	Gas optimization suggestions	Informational	Gas Optimization	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. User funds may be locked in the contract	
Severity: High	Category: Business Logic
Target: - contracts/KToken.sol	

Description

Unlike the original ERC4626 contract, in the KToken contract, the share price is based on the accRewardsPerToken and accPnlPerToken variables. The accRewardsPerToken variable can only be updated through the distributeReward() function, in which users can donate assets. The accPnlPerToken variable is related to the part of assets transferred in and out. Simply transferring tokens to the contract can not update these two variables.

Thus, if a user deposits some assets through the deposit() function and the calculated share amount is 0, his funds will be locked in the contract forever.

contracts/KToken.sol:L142-L146

```
function deposit(uint assets, address receiver) public override checks(assets) returns
(uint) {
    uint shares = previewDeposit(assets);
    _deposit(_msgSender(), receiver, assets, shares);
    return shares;
}
```

Recommendation

Consider reverting the transaction if the calculated amount of share is 0.

2. Cooldown mechanism may be bypassed via token transfer	
Severity: Medium	Category: Business Logic
Target: - contracts/KToken.sol	

Description

Users can request a withdrawal through the makeWithdrawRequest() or makeRedeemRequest() functions and settle the withdrawal after the cool period. However, shares that have been requested to withdraw will not be recorded or locked. Thus, users can make a request every epoch with the same shares.

contracts/KToken.sol:L154-L162

```
function makeWithdrawRequest(uint assets) public checks(assets) {
    require(assets <= maxWithdraw(_msgSender()), "ERC4626: withdraw more than max");

    (uint256 currentEpoch,) = this.updateEpoch();
    uint shares = previewWithdraw(assets);
    reqWithdrawals[_msgSender()][currentEpoch+coolPeriod] = shares;

    emit WithdrawRequested(_msgSender(), assets);
}
```

contract/KToken.sol:L172-L180

```
function withdraw(uint assets, address receiver, address _owner) public override
checks(assets) returns (uint) {
    uint shares = previewWithdraw(assets);
    (uint256 currentEpoch,) = this.updateEpoch();
    require(reqWithdrawals[_msgSender()][currentEpoch] == shares, "request first");
    reqWithdrawals[_msgSender()][currentEpoch] = 0;

    _withdraw(_msgSender(), receiver, _owner, assets, shares);
    return shares;
}
```

Here is a possible scenario (the cool period lasts for 2 epochs):

1. Alice owns 1 share and makes withdrawal requests for 1 share at epoch 1 and 2;
2. Alice settles the withdrawal for 1 share at epoch 3;
3. Bob also wants to withdraw 1 share and transfer his share to Alice. Alice withdraws Bob's share at epoch 4. Thus, Bob's cool period is bypassed.

Recommendation

Consider recording or locking shares that have been requested to withdraw.

3. Chainlink’s latestRoundData() might return stale results	
Severity: Medium	Category: Data Validation
Target: - contracts/PriceAggregator.sol	

Description

In the fulfill() function, the price provided by the node may be compared with the return value of the Chainlink’s latestRoundData(). According to Chainlink’s documentation, the answer will be updated when the value deviates beyond a specified threshold or when the heartbeat idle time has passed. However, if answers are not updated in time, prices provided by nodes may be compared with stale prices, affecting related trades.

contracts/PriceAggregator.sol:L149-L179

```
function fulfill(bytes32 _requestId, uint _price) external
recordChainlinkFulfillment(_requestId){
        ...
        uint feedPrice = _price;

        IPairsStorage.Feed memory f = pairsStorage.pairFeed(r.pairIndex);
        if (f.feed1 != address(0)) {
                (, int feedPrice1, , , ) = IChainlinkFeed(f.feed1).latestRoundData();

            if(f.feedCalculation == IPairsStorage.FeedCalculation.DEFAULT){
                feedPrice = uint(feedPrice1*int(PRECISION)/1e8);
            }else if(f.feedCalculation == IPairsStorage.FeedCalculation.INVERT){
                feedPrice = uint(int(PRECISION)*1e8/feedPrice1);
            }else{
                (, int feedPrice2, , , ) = IChainlinkFeed(f.feed2).latestRoundData();
                feedPrice = uint(feedPrice1*int(PRECISION)/feedPrice2);
            }
        }

        uint priceDiff = _price >= feedPrice ? (_price - feedPrice) : (feedPrice -
_price);
        if(priceDiff * PRECISION * 100 / feedPrice <= f.maxDeviationP)
```

Recommendation

It is recommended to track the updatedAt value from the latestRoundData() function to make sure that the answer is recent enough to be compared. If the reported answer is not updated within the heartbeat or within the acceptable time limits, consider rejecting it.

4. Setting maxOpenInterestDai to 0 will not pause trading on a specific pair	
Severity: Medium	Category: Business Logic
Target: - contracts/TradingStorage.sol - contracts/TradingCallbacks.sol	

Description

Setting maxOpenInterestDai to 0 for a specific pair is expected to pause trading on that pair.

contracts/TradingStorage.sol:L207-L211

```
function setMaxOpenInterestDai(uint _pairIndex, uint _newMaxOpenInterest) external
onlyGov{
        // Can set max open interest to 0 to pause trading on this pair only
        openInterestDai[_pairIndex][2] = _newMaxOpenInterest;
        ...
}
```

While in the TradingCallbacks contract, the withinExposureLimits() function could return true if maxOpenInterestDai is 0.

contracts/TradingCallbacks.sol:L445-L451

```
function withinExposureLimits(uint _pairIndex, bool _buy, uint _positionSizeDai, uint
_leverage) private view returns(bool){
        ...
        uint256 maxOpenInterestDai = storageT.openInterestDai(_pairIndex, 2);
        return (maxOpenInterestDai == 0 || storageT.openInterestDai(_pairIndex, _buy ? 0
: 1) + _positionSizeDai * _leverage <= maxOpenInterestDai)
        && pairsStored.groupCollateral(_pairIndex, _buy) + _positionSizeDai <=
pairsStored.groupMaxCollateral(_pairIndex);
}
```

Consider withinExposureLimits() function’s usage in the openTradeMarketCallback() and executeNftOpenOrderCallback() functions, it should instead return false if maxOpenInterestDai is 0 to pause the trading.

Recommendation

It is recommended to let the withinExposureLimits() function return false if maxOpenInterestDai is 0.

5. Lack of duplicate checks for nodes

Severity: Low	Category: Data Validation
Target: - contracts/PriceAggregator.sol	

Description

In the PriceAggregator contract, the addNode() function implements duplicate checks for input nodes but there are no node duplicate checks in the initialize() and replaceNode() functions.

contracts/PriceAggregator.sol:L61-L76

```
function initialize(
        IStorage _storageT,
        IPairsStorage _pairsStorage,
        address[] memory _nodes,
        address _linkAddress, address _triggerInfo
) external initializer {
        require(_nodes.length > 0, "WRONG_PARAMS");
        ...
        nodes = _nodes;
        ...
}
```

contracts/PriceAggregator.sol:L98-L105

```
function replaceNode(uint _index, address _a) external onlyGov{
    require(_index < nodes.length, "WRONG_INDEX");
    require(_a != address(0), "VALUE_0");

    emit NodeReplaced(_index, nodes[_index], _a);

    nodes[_index] = _a;
}
```

Recommendation

It is recommended to also implement duplicate checks for input nodes in the initialize() and replaceNode() functions.

6. Improper LINK fee for NFT order execution

Severity: Low	Category: Business Logic
Target: - contracts/Trading.sol - contracts/TradingStorage.sol	

Description

Triggers need to pay LINK fees for every NFT order execution.

contracts/Trading.sol:L375-L382

```
function executeNftOrder(
    IStorage.LimitOrder _orderType,
    address _trader,
    uint _pairIndex,
    uint _index
) external notContract notDone{
    ...
    // link transfer
    storageT.transferLinkToAggregator(msg.sender);
    uint orderId = aggregator.getPrice(
        _pairIndex,
        _orderType == IStorage.LimitOrder.OPEN ?
            IAggregator.OrderType.LIMIT_OPEN :
            IAggregator.OrderType.LIMIT_CLOSE
    );
    ...
}
```

The actual LINK fee spent in a single price request is linkFee() * nodes.length.

contracts/PriceAggregator.sol:L132-L143

```
function getPrice(
       uint _pairIndex,
       OrderType _orderType
) external onlyTrading returns(uint){
    ...
    uint linkFeePerNode = linkFee();
    ...
    for(uint i = 0; i < nodes.length; i ++){
       orderIdByRequest[sendChainlinkRequestTo(nodes[i], linkRequest, linkFeePerNode)] =
orderId;
    }
```

However, triggers only pay linkFee() for one node.

contracts/TradingStorage.sol:L391-L393

```
function transferLinkToAggregator(address _from) external onlyTrading{
       linkErc677.transferFrom(_from, address(priceAggregator),
priceAggregator.linkFee());
}
```

Recommendation

Consider updating the logic in the transferLinkToAggregator() function so that triggers pay for enough LINK.

7. Inaccurate funding/rollover fee calculation	
Severity: Low	Category: Numerics
Target: - contracts/TradingCallbacks.sol - contracts/PairInfo.sol - contracts/Trading.sol	

Description

When users close trades, the contract will calculate related funding fees and rollover fees based on positionSizeDai.

contracts/TradingCallbacks.sol:L364-L381

```
function unregisterTrade(
    IStorage.Trade memory _trade,
    int _percentProfit,     // PRECISION
    uint _lpFee,            // 1e18
    uint _triggerReward,    // 1e18
    address _trigger
) private returns(uint daiSentToTrader){
    // 1. Calculate net PnL (before closing and holding fees)
    daiSentToTrader = pairInfos.getTradeValue(
        _trade.trader,
        _trade.pairIndex,
        _trade.index,
        _trade.buy,
        _trade.positionSizeDai,
        _trade.leverage,
        _percentProfit,
        0
    );
```

contracts/PairInfos.sol:L445-L463

```
function getTradeValue(
        address trader,
        uint pairIndex,
        uint index,
        bool long,
        uint collateral,   // 1e18 (DAI)
        ...
) external onlyCallbacks returns(uint amount){ // 1e18 (DAI)
        storeAccFundingFees(pairIndex);

        uint r = getTradeRolloverFee(trader, pairIndex, index, collateral);
        int f = getTradeFundingFee(trader, pairIndex, index, long, collateral, leverage);

        ...
}
```

The positionSizeDai could be changed in the updateSl() function. If the positionSizeDai becomes smaller, the rollover fee paid by users will decrease, and the funding fees will also decrease or increase.

contracts/Trading.sol:L299-L307

```
function updateSl(uint _pairIndex, uint _index, uint _newSl) external notDone {
    ...
    uint levPosDai = t.initialPosToken * t.leverage;

    t.positionSizeDai -= storageT.handleDevGovFees(
        t.pairIndex,
        levPosDai / 2
    );

    storageT.updateTrade(t);
    ...
}
```

Recommendation

Consider calculating accumulated rollover fees and funding fees in a timely manner each time positionSizeDai changes.

8. Lack of input parameter checks in the setPairParams() function	
Severity: Low	Category: Data Validation
Target: - contracts/PairInfos.sol	

Description

If setting through the setRolloverFeePerBlockP() and setFundingFeePerBlockP() functions, there are checks to ensure that the input value does not exceed a cap value.

contracts/PairInfos.sol:L198-L206

```
function setFundingFeePerBlockP(uint pairIndex, uint value) public onlyManager{
        require(value <= 10000000, "TOO_HIGH"); // ≈ 40% per day

        storeAccFundingFees(pairIndex);

        pairParams[pairIndex].fundingFeePerBlockP = value;

        emit FundingFeePerBlockPUpdated(pairIndex, value);
}
```

contracts/PairInfos.sol:L177-L185

```
function setRolloverFeePerBlockP(uint pairIndex, uint value) public onlyManager{
        require(value <= 25000000, "TOO_HIGH"); // ≈ 100% per day

        storeAccRolloverFees(pairIndex);

        pairParams[pairIndex].rolloverFeePerBlockP = value;

        emit RolloverFeePerBlockPUpdated(pairIndex, value);
}
```

However, there is no such check for RolloverFeePerBlockP and FundingFeePerBlockP in the setPairParams() function.

contracts/PairInfos.sol:L131-L138

```
function setPairParams(uint pairIndex, PairParams memory value) public onlyManager{
        storeAccRolloverFees(pairIndex);
        storeAccFundingFees(pairIndex);

        pairParams[pairIndex] = value;

        emit PairParamsUpdated(pairIndex, value);
}
```

Recommendation

It is recommended to implement similar checks for RolloverFeePerBlockP and FundingFeePerBlockP in the setPairParams() function.

9. Unimplemented functions	
Severity: Low	Category: Business Logic
Target: - contracts/Trading.sol - contracts/TradingCallbacks.sol	

Description

There are some functions called using interfaces, but no implemented functions can be found in the relevant contracts of the current code base, e.g. IAggregator.triggerInfo().

contracts/TradingCallbacks.sol:L247-L248

```
IAggregator aggregator = storageT.priceAggregator();
ITriggerInfo triggerInfo = aggregator.triggerInfo();
```

Recommendation

It is recommended to inherit the corresponding interface when implementing the contract to avoid omissions.

2.3 Informational Findings

10. Share price changes may cause the call to withdraw() to fail	
Severity: Informational	Category: Business Logic
Target: - contracts/KToken.sol	

Description

Users can request a withdrawal for a specific amount of assets through the makeWithdrawRequest() function. The makeWithdrawRequest() will record the corresponding share amounts. After the cool period epoch, the withdrawal request can be settled through the withdraw() or redeem() functions.

contracts/KToken.sol:L154-L162

```
function makeWithdrawRequest(uint assets) public checks(assets) {
        ...
        uint shares = previewWithdraw(assets);
        reqWithdrawals[_msgSender()][currentEpoch+coolPeriod] = shares;
        ...
}
```

However, the share price may change during the cool period, causing the return value of the previewWithdraw() function to be different between makeWithdrawRequest() and withdraw(). Therefore, the calculated amount of shares may not match the recorded amount, resulting in revert. In this case, users can only use the redeem() function to withdraw.

contracts/KToken.sol:L172-L180

```
function withdraw(uint assets, address receiver, address _owner) public override
checks(assets) returns (uint) {
        uint shares = previewWithdraw(assets);
        (uint256 currentEpoch,) = this.updateEpoch();
        require(reqWithdrawals[_msgSender()][currentEpoch] == shares, "request first");
        reqWithdrawals[_msgSender()][currentEpoch] = 0;

        _withdraw(_msgSender(), receiver, _owner, assets, shares);
        return shares;
}
```

contracts/KToken.sol:L182-L190

```
function redeem(uint shares, address receiver, address _owner) public override
checks(shares) returns (uint) {
        (uint256 currentEpoch,) = this.updateEpoch();
        require(reqWithdrawals[_msgSender()][currentEpoch] == shares, "request first");
        reqWithdrawals[_msgSender()][currentEpoch] = 0;

        uint assets = previewRedeem(shares);
        _withdraw(_msgSender(), receiver, _owner, assets, shares);
        return assets;
}
```

Recommendation

It is recommended to inform users about the possibility of failure to withdraw using the withdraw() function in the document.

11. Inconsistency between documentation and implementation	
Severity: Informational	Category: Inconsistency
Target: - contracts/PairInfos.sol	

Description

The document states that liquidation price distance is calculated according to the following formula:

Liquidation Price Distance = Open Price * (Collateral * 0.9 - Funding Fee) / Collateral / Leverage.

However, the calculation of liquidation price distance in the code is different from what is stated in the document.

contracts/PairInfos.sol:L432-L435

```
int liqPriceDistance = int(openPrice) * (
       int(collateral * LIQ_THRESHOLD_P / 100)
       - int(rolloverFee) - fundingFee
    ) / int(collateral) / int(leverage);
```

Recommendation

Consider fixing the mismatch between the documentation and implementation.

12. Redundant code	
Severity: Informational	Category: Redundancy
Target: - contracts/Trading.sol - contracts/TriggerInfo.sol - contracts/PriceAggregator.sol - contracts/TradingCallbacks.sol - contracts/TradingStorage.sol	

Description

The following events are defined but not used.

contracts/Trading.sol:L34,L39

```
event AddressUpdated(string name, address a);
event TriggerOrderSameBlock(address nftHolder, address trader, uint pairIndex);
```

contracts/TriggerInfo.sol:L36

```
event TokensClaimed(address bot, uint tokens);
```

contracts/PriceAggregator.sol:L40

```
event AddressUpdated(string name, address a);
```

contracts/TradingCallbacks.sol:L66-L67

```
event AddressUpdated(string name, address a);
event NumberUpdated(string name, uint value);
```

Since the storeOpenLimitOrder() function in the TradingStorage contract will also calls the firstEmptyOpenLimitIndex() function, the index query in the Trading contract can be safely removed.

contracts/TradingStorage.sol:L300-L301

```
function storeOpenLimitOrder(OpenLimitOrder memory o) external onlyTrading{
    o.index = firstEmptyOpenLimitIndex(o.trader, o.pairIndex);
```

contracts/Trading.sol:L138-L156

```
uint index = storageT.firstEmptyOpenLimitIndex(msg.sender, t.pairIndex);

storageT.storeOpenLimitOrder(
        IStorage.OpenLimitOrder(
            msg.sender,
            t.pairIndex,
            index,
            t.positionSizeDai,
            ...
        )
);
```

Recommendation

Consider removing the redundant code.

13. Gas optimization suggestions	
Severity: Informational	Category: Gas Optimization
Target: - contracts/KToken.sol - contracts/TradingCallbacks.sol - contracts/PriceAggregator.sol	

Description

contracts/KToken.sol:L232-L240

```
function updateEpoch() external returns (uint256, uint256) {
       epochDuration = 20 minutes;
       ...
}
```

Each time the updateEpoch() function is called, the epochDuration variable is set to 20 minutes. Since it is not used in other functions, consider declaring it as a constant.

contracts/TradingCallbacks.sol:L201-L203

```
if((t == ITriggerInfo.OpenLimitOrderType.LEGACY ? (a.price >= o.minPrice && a.price <=
o.maxPrice) :
    t == ITriggerInfo.OpenLimitOrderType.REVERSAL ? (o.buy ? a.price <= o.maxPrice :
a.price >= o.minPrice) :
    (o.buy ? a.price >= o.minPrice : a.price <= o.maxPrice))
```

If the order type is LEGACY, the setOpenLimitOrderType() function will not be called. Thus, the TriggerInfo contract will not store LEGACY orders and the above code can be simplified to:

```
if((t == ITriggerInfo.OpenLimitOrderType.REVERSAL ? (o.buy ? a.price <= o.maxPrice :
a.price >= o.minPrice) :
(o.buy ? a.price >= o.minPrice : a.price <= o.maxPrice))
```

contracts/PriceAggregator.sol:L149-L157

```
function fulfill(bytes32 _requestId, uint _price) external
recordChainlinkFulfillment(_requestId){
    if (_price == 0) {
        return;
    }
    uint orderId = orderIdByRequest[_requestId];
    Order storage r = orders[orderId];
    delete orderIdByRequest[_requestId];
```

If _price is equal to 0, it is recommended to delete orderIdByRequest[_requestId] to save gas.

contracts/Trading.sol:L92-L191

```
function openTrade(
       ...
       uint _spreadReductionId,
       ...
) external notDone {
       ...
```

SALUS

```
        uint spreadReductionP = _spreadReductionId > 0 ?
storageT.spreadReductionsP(_spreadReductionId-1) : 0;
        ...
        // legacy
        require(_spreadReductionId == 0, "NO_CORRESPONDING_NFT_SPREAD_REDUCTION");
        ...
}
```

In the openTrade() function, _spreadReductionId is required to be 0, so spreadReductionP can be set to 0 directly or consider removing the _spreadReductionId parameter.

Recommendation

Consider making changes based on the above suggestions.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 98ffa7c:

File	SHA-1 hash
contracts/KToken.sol	c83eb812643aecfba9da7f9a3507aa28c4dba32b
contracts/PairInfos.sol	b1c72a489a7b2c92f6cc2b536cf6b99ba42dfb6a
contracts/PairStorage.sol	7d08796c7fc184f6ce3fbb3471165f40b0c48ec0
contracts/PriceAggregator.sol	2b7ab92c68f1a687cf8036238133573fb55a7af9
contracts/Trading.sol	73241095dc81e85d3901f1a5526c27ccb7198612
contracts/TradingCallbacks.sol	d42ec6a11615ac9c90f06523886ee4f0e07fbebb
contracts/TradingStorage.sol	224c26adf40a3f834ad5562a20f9cf34808c85dd
contracts/TriggerInfo.sol	22e906f56da80d4c4cb9dc69bb5c98114a0222d7